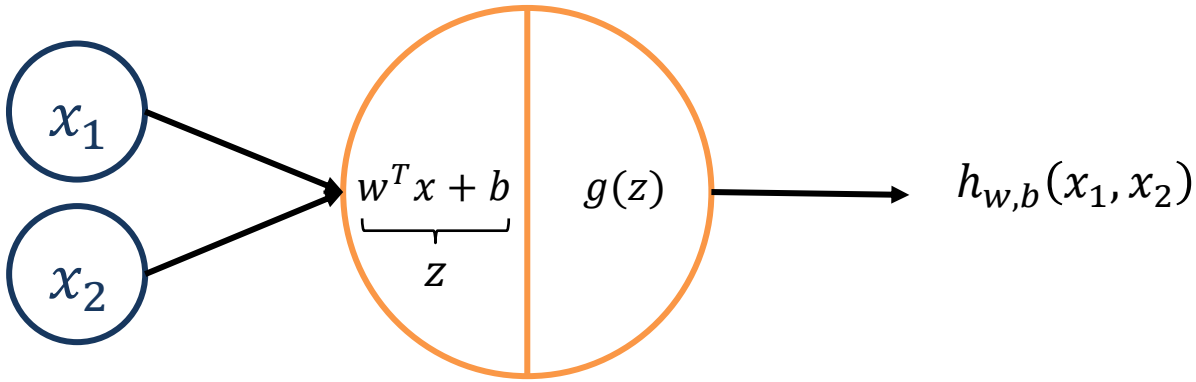
Neural Networks

Prof. Giuseppe Serra

Neural Network Representation

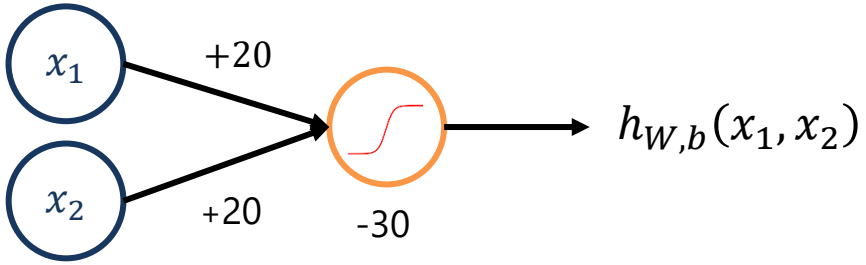
This network means that to compute the value of the decision function, we need to multiply first input x_1 with θ_1, second input x_2 with θ_2, then add two values and b, then apply the sigmoid function



Simple Neural Networks

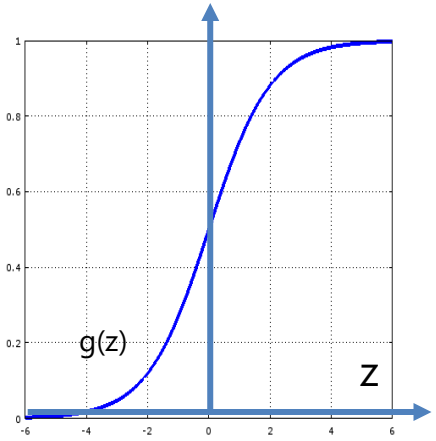
Example: AND Logic Port

$$y = x_1 \text{ AND } x_2$$



$$h_{w,b}(x_1, x_2) = g(-30 + 20x_1 + 20x_2)$$

g is the Sigmoid Function



x_1	x_2	$h_{w,b}(x_1, x_2)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

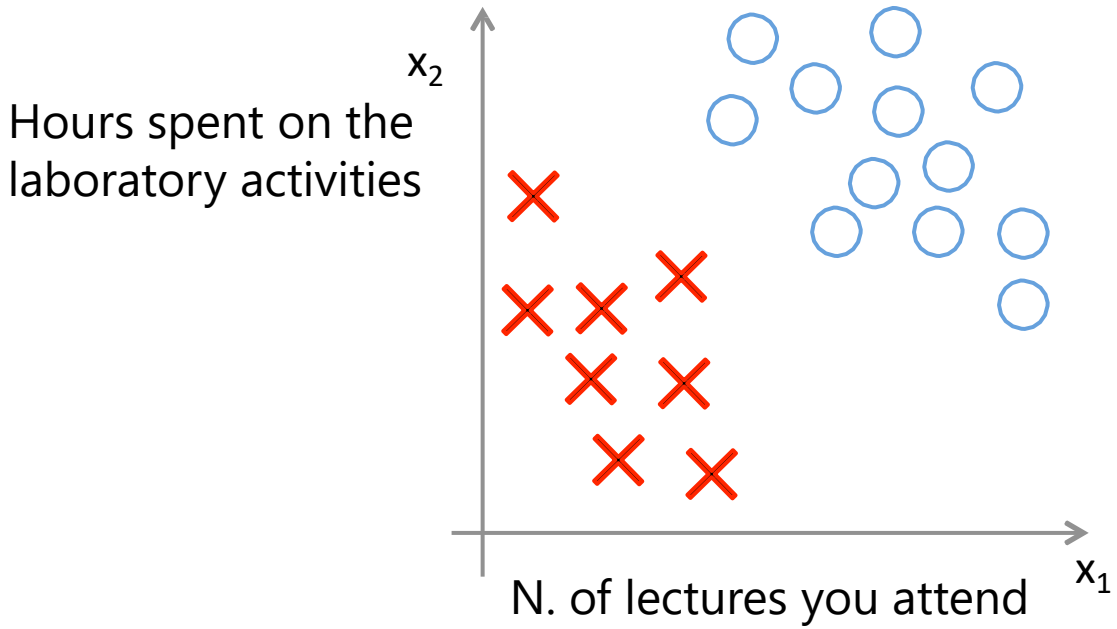
Example Problem: Will I Pass this Class?

Let's start with a simple two feature model:

x_1 = Number of lectures you attend

x_2 = Hours spent on the laboratory's activities

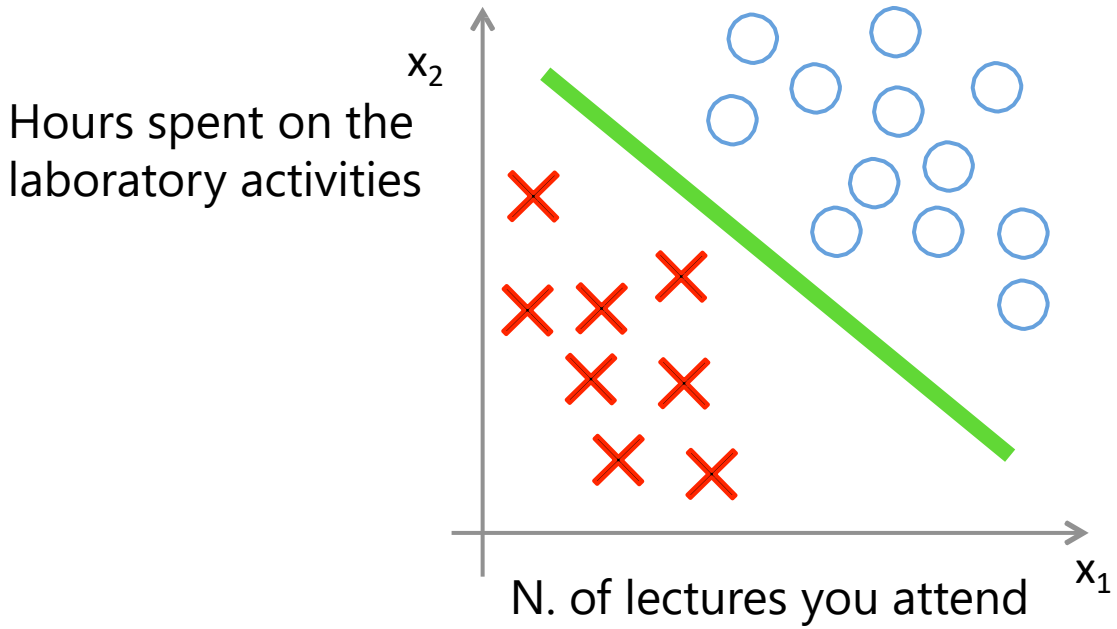
Example Problem: Will I Pass this Class?



O corresponds to “Pass”

X corresponds to “Fail”

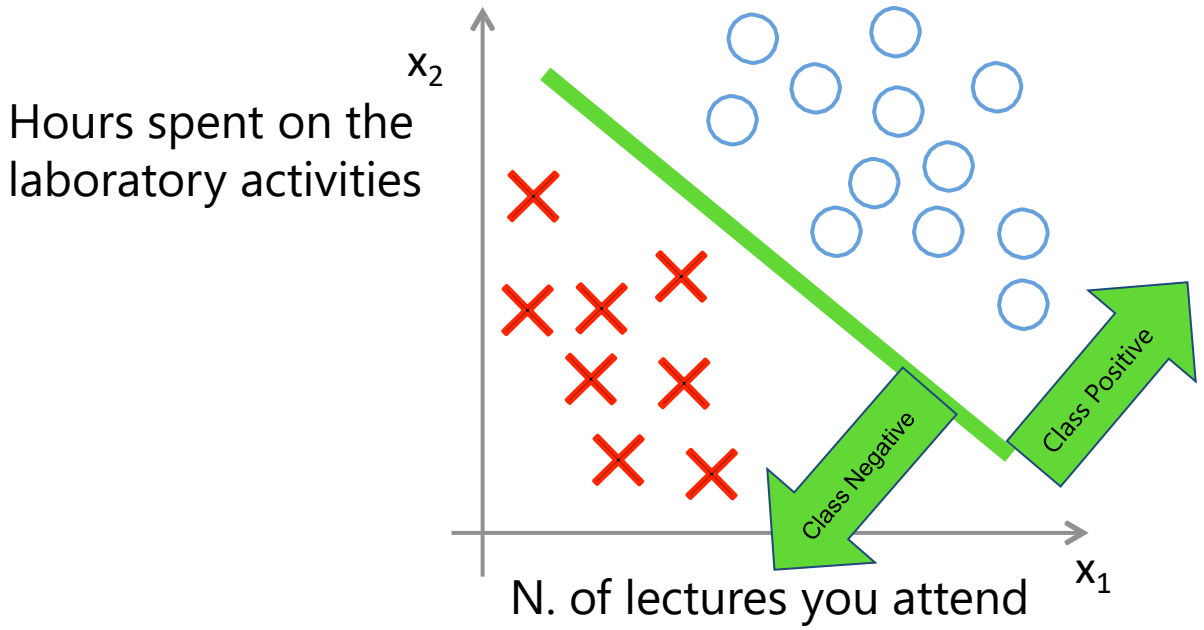
Example Problem: Will I Pass this Class?



O corresponds to “Pass”

X corresponds to “Fail”

Example Problem: Will I Pass this Class?

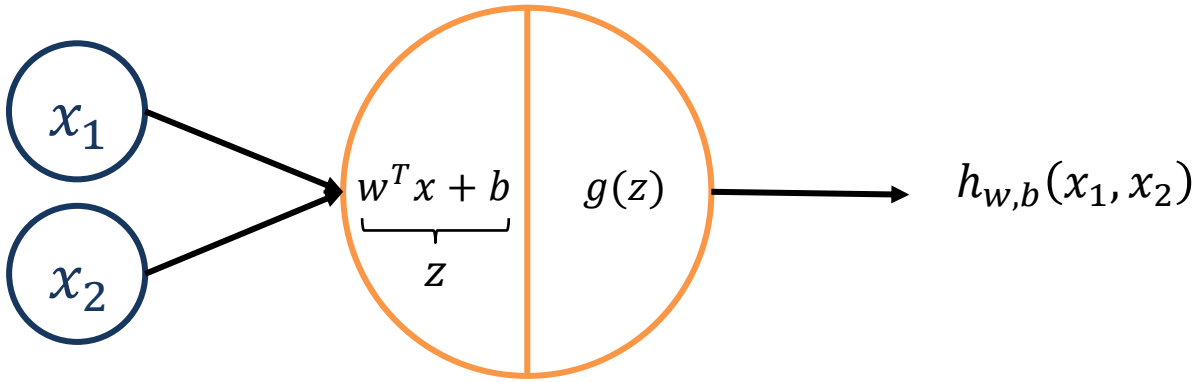


O corresponds to “Pass”

X corresponds to “Fail”

Neural Network with One Hidden Layer

Let's try to a simple Neural Network



A neural network with only one hidden layer is considered a shallow neural network.

Let's try by yourself: https://phiresky.github.io/neural-network-demo/?preset=Binary%20Classifier%20for%20XOR

Neural Network

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ \ldots, (x^{(m)}, y^{(m)}),\}$

In our case $x^{(i)}$ is equal to the i-th feature and $y^{(i)}$ is the i-th label (pass/fail)

With a Neural Network we want to learn a probabilistic function that $\hat{y} = P(y = 1|x)$

In particular, **the goal is to find the parameters w and b** of the following function (hypothesis):

$h_{w,b}(x) = g\left(w^T x + b\right) = \frac{1}{1+e^{-(w^T x+b)}}$, where $g(z)$ is the Sigmoid function,

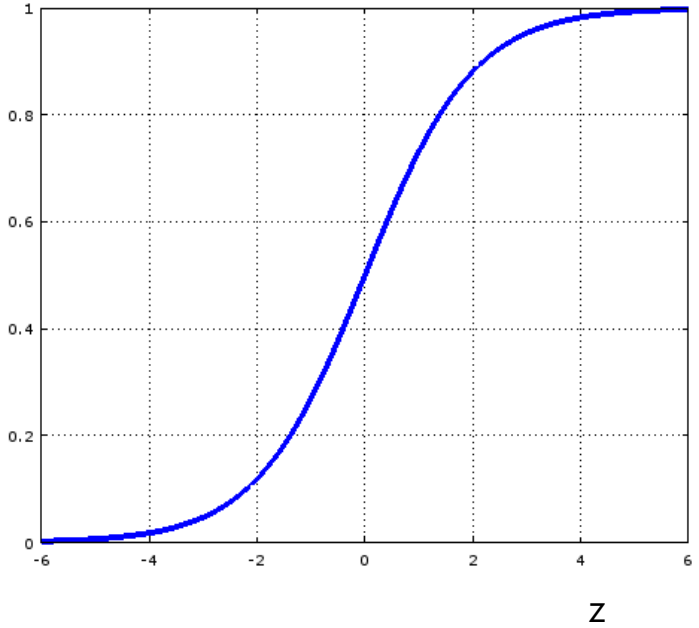
so that $\begin{cases} h_{w,b}(x) \geq 0.5 & if \ \ y = 1 \\ h_{w,b}(x) < 0.5 & if \ \ y = 0 \end{cases}$

To get our discrete 0 or 1 classification, we map the output of the hypothesis function as follows:

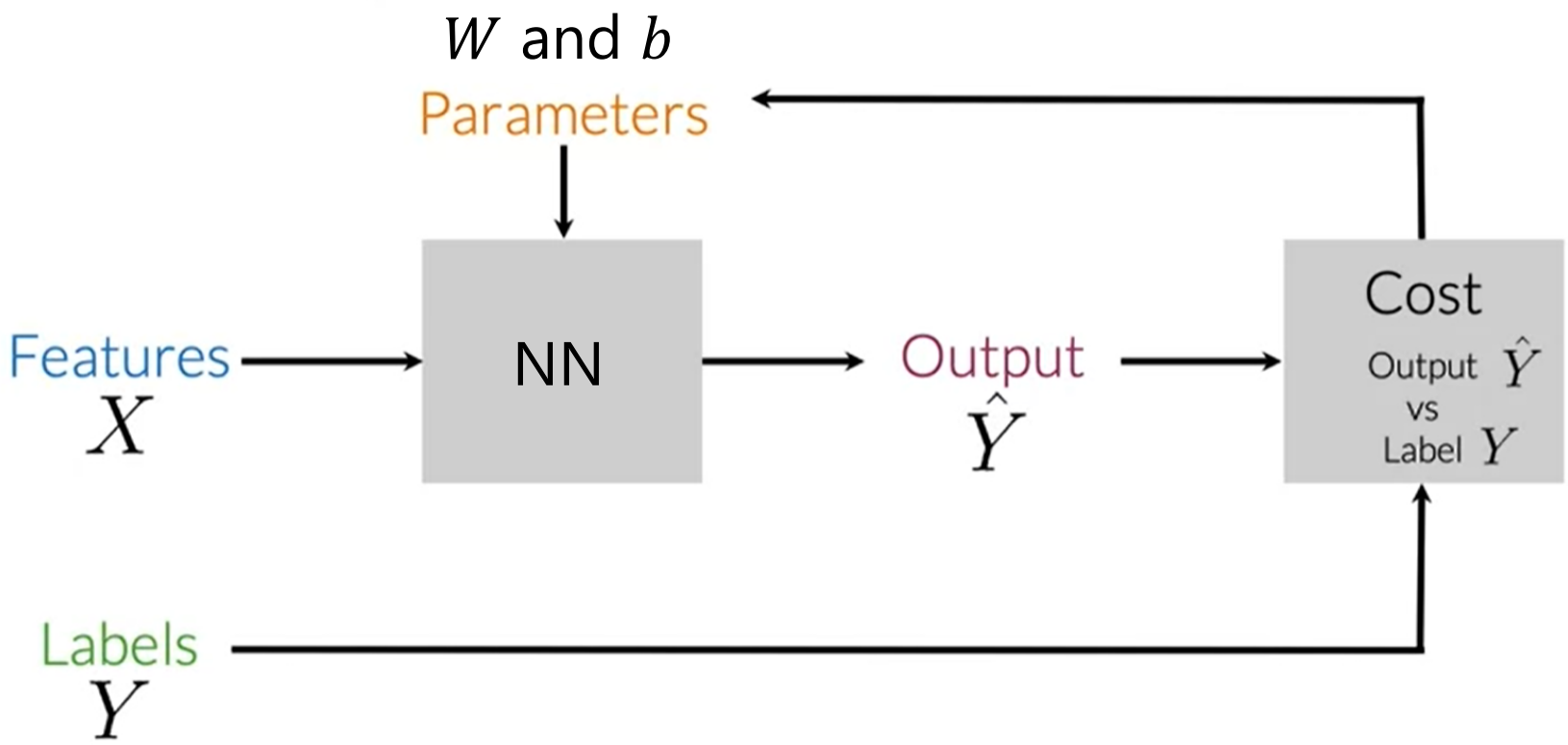
$h_{w,b}(x) \geq 0.5 \ \ \rightarrow$ " 1"

$h_{w,b}(x) < 0.5 \ \rightarrow$ " 0"

Sigmoid Function



Classification: Neural Network

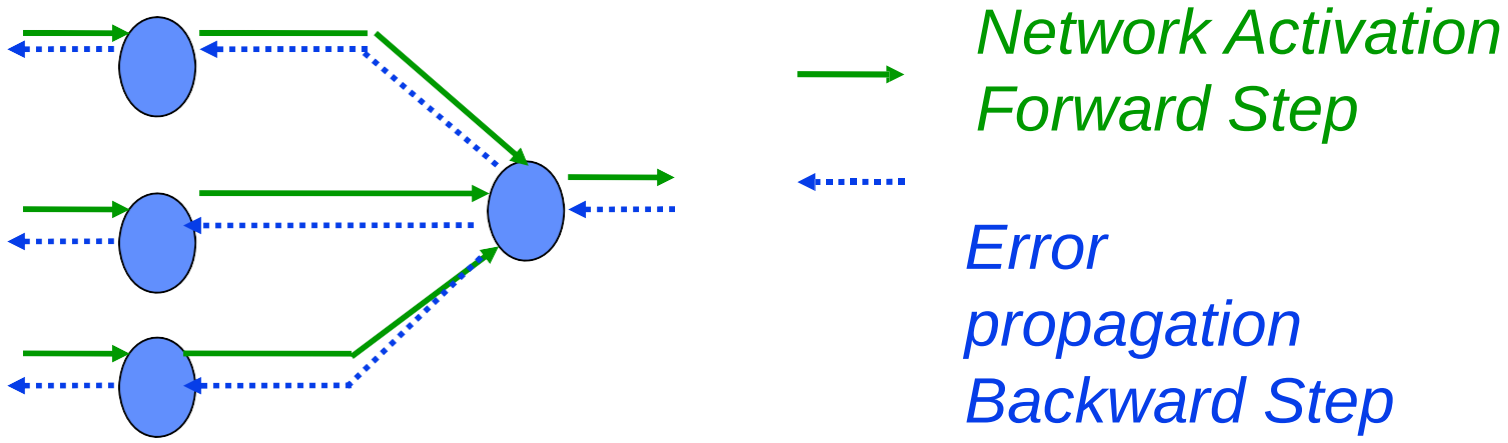


Training a Neural Networks (computing weights)

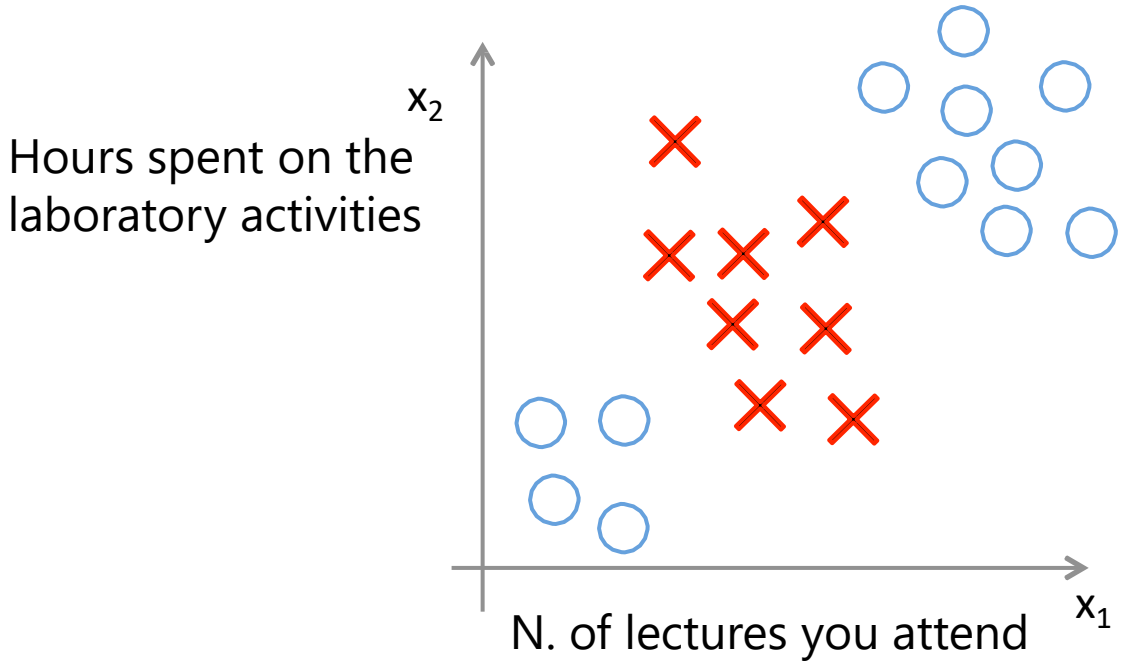
Random initialization of the weights

Backpropagation consists of the repeated application of the following two steps:

- Forward step: in this step the network is activated on one example and the error of (each neuron of) the output layer is computed.
- Backward step: in this step the network error is used for updating the weights. Starting at the output layer, the error is propagated backwards through the network, layer by layer. **This is done by recursively computing the local gradient of each neuron**.



A Complex Problem

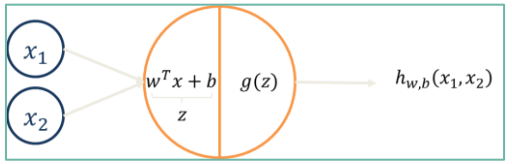


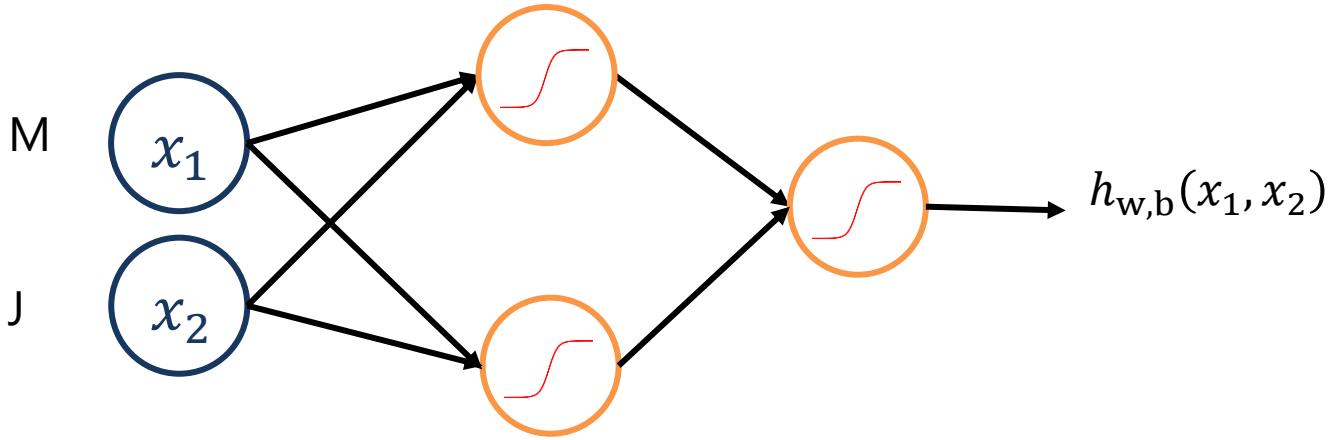
O corresponds to “Pass”

X corresponds to “Fail”

Neural Network with One Hidden Layer

Let's try to a Neural Network with one hidden layer!





Input Layer Hidden Layer Output Layer

Let's try by yourself: https://phiresky.github.io/neural-network-demo/?preset=Binary%20Classifier%20for%20XOR

Hyperparameters

K layers = depth of network

D_k hidden units per layer = width of network

These are called hyperparameters – chosen before training the network

Can try retraining with different hyperparameters – hyperparameter optimization or hyperparameter search

Activation Functions

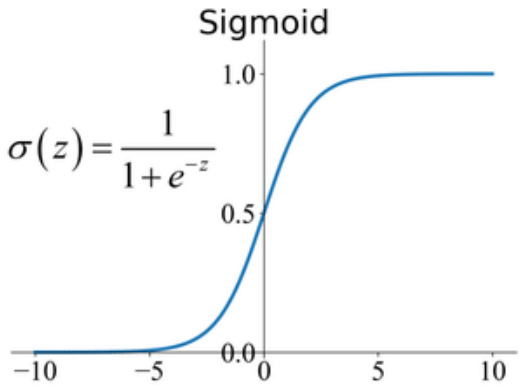
Activation Functions

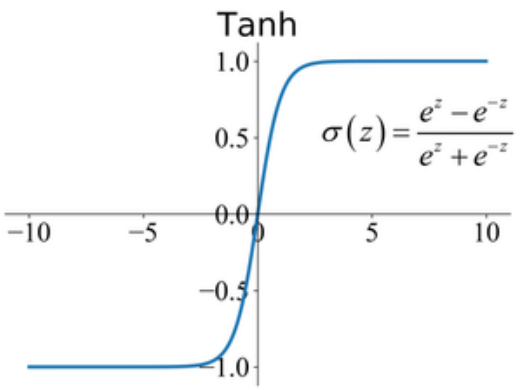
Sigmoid Function:

- it has seen frequent use historically, yet now it is rarely used in the intermediate layers, because it kills the gradient (remember $\frac{d\,\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$).
- It is commonly use in the last layer is you want a prediction value between [0,1]

Tanh Function:

- "…the hyperbolic tangent activation function typically performs better than the logistic sigmoid." – Page 195, Deep Learning, 2016
- $\frac{d\,tanh(z)}{dz} = \ldots = 1 - tanh^2(z)$





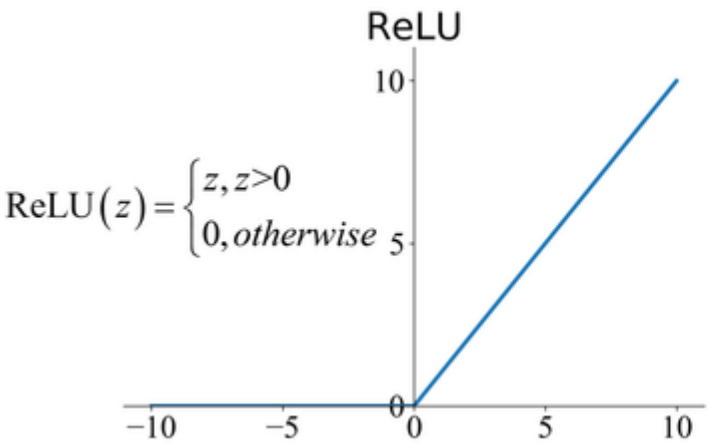
Activation Functions

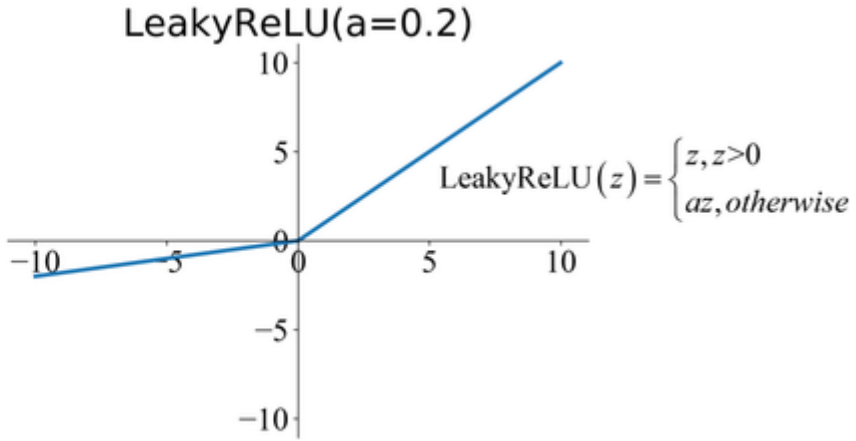
Rectified linear unit (ReLU):

- It has become the **default activation function** for many types of neural networks, because a model that uses **it is easier to train** and often achieves better performance. The derivative of the rectified linear function is also **easy to calculate**. **The slope for negative values is 0.0 and the slope for positive values is 1.0. Derivative in zero is not defined** (empirically is not a problem)

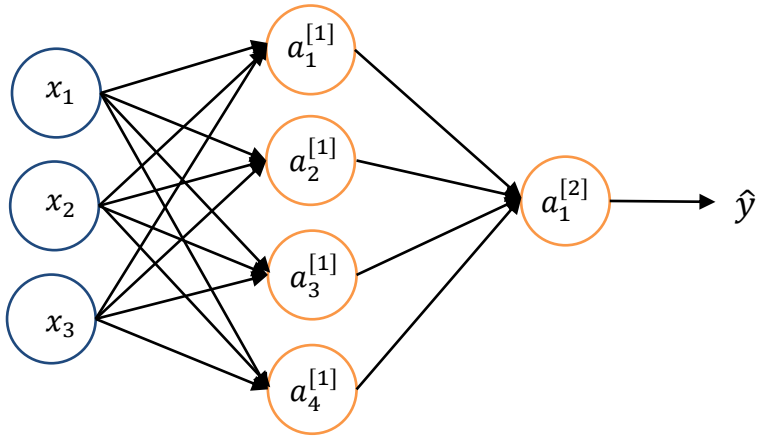
Leaky version of a Rectified Linear Unit (LeakyReLU)

- ReLU is affect to the **Dying ReLU Problem**: some **ReLU Neurons essentially die** for all inputs and remain inactive no matter what input is supplied, here no gradient flows (**the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn**). **Empirically performance is similar to the ReLU**.





Neural Network Representation



$W^{[j]}$ = matrix of weights controlling function mapping from layer $j-1$ to layer j
$b^{[j]}$ = bias
$a^{[j]}$ = activation of vector for layer j
$g^{[j]}$ = activation function for layer j

Given in input X:

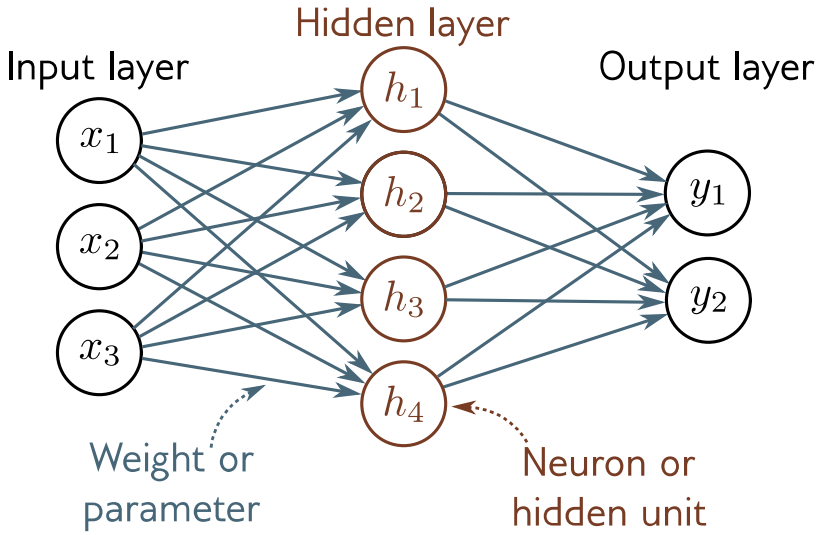
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
(4,1) (4,3) (3,1) (4,1) - Dimensionality

$$a^{[1]} = g^{[1]}(z^{[1]})$$
(4,1) (4,1) - Dimensionality

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
(1,1) (1,4) (4,1) (1,1) - Dimensionality

$$a^{[2]} = g^{[2]}(z^{[2]})$$
(1,1) (1,1) - Dimensionality

Terminology



- Everything in one layer connected to everything in the next = fully connected network
- No loops = feedforward network
- Values after activation functions = activations
- Values before activation functions = pre-activations
- One hidden layer = shallow neural network
- More than one hidden layer = deep neural network
- Number of hidden units ≈ capacity

Why do you need non-linear functions?

Given in input X:

$z^{[1]} = W^{[1]}x + b^{[1]}$

(4,1) (4,3) (3,1) (4,1) - Dimensionality

$a^{[1]} = g^{[1]}(z^{[1]})$

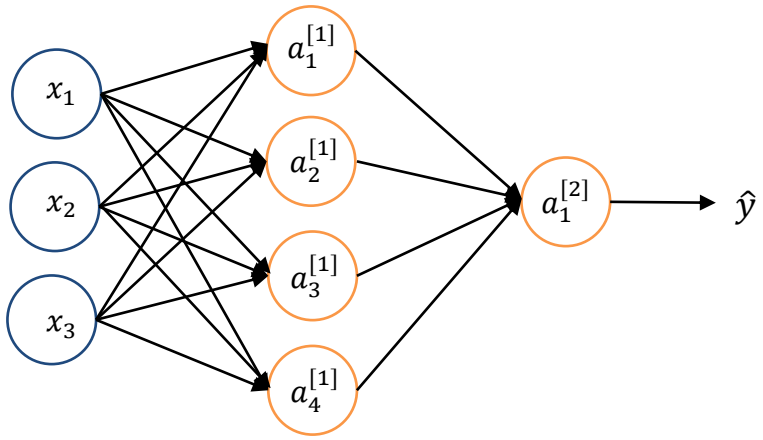
(4,1) (4,1) - Dimensionality

$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

(1,1) (1,4) (4,1) (1,1) - Dimensionality

$a^{[2]} = g^{[2]}(z^{[2]})$

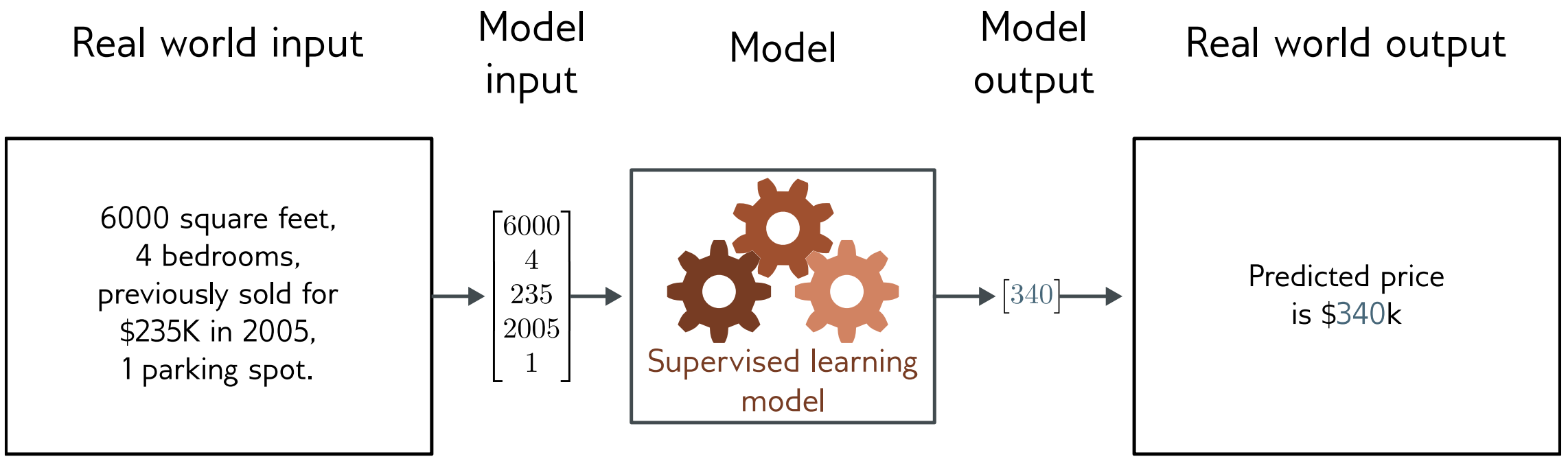
(1,1) (1,1) - Dimensionality



- Let's suppose $g^{[1]}(z) = g^{[2]}(z) = z$ (identity function or linear activation function) then:
- $a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} = W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} = (W^{[2]}W^{[1]})x + (W^{[2]}b^{[1]} + b^{[2]})$
- Let's define $W' = W^{[2]}W^{[1]}$ and $b' = (W^{[2]}b^{[1]} + b^{[2]})$
- $a^{[2]} = W'x + b'$ (Thus, if you are using identity activation functions, the output neural networks is linear combination of the input)
- Linear activation is often useless, unless in the last layer of a neural network when you are working in a regression task.
- Let's try again by yourself (use linear activation functions): https://phiresky.github.io/neural-network-demo/?preset=Binary%20Classifier%20for%20XOR

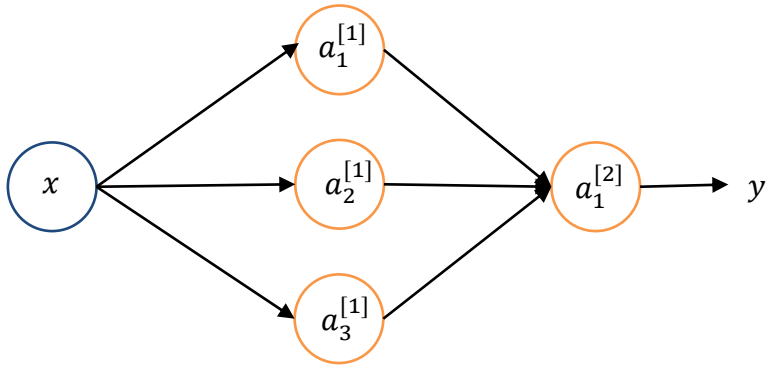
Regression

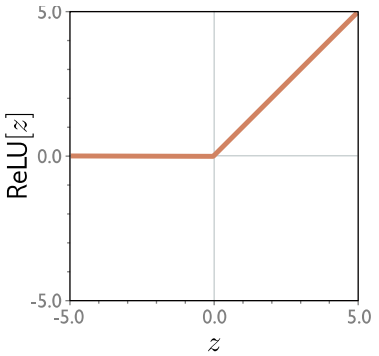
Regression


Real world input
Model input
Model
Model output
Real world output
6000 square feet,
4 bedrooms,
previously sold for
$235K in 2005,
1 parking spot.
6000
4
235
2005
1
Supervised learning model
340
Predicted price
is $340k

Example Shallow Networks

Let' consider the following shallow network





Activation function

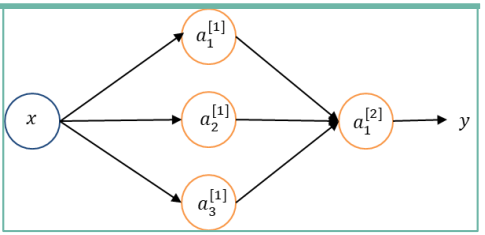
$$y = \mathrm{f}[x, \boldsymbol{\phi}]$$
$$= \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x]$$

$$\mathrm{a}[z] = \mathrm{ReLU}[z] = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}.$$

Rectified Linear Unit

(particular kind of activation function)

Example Shallow Networks



$$y = \mathrm{f}[x, \boldsymbol{\phi}]$$
$$= \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11} x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21} x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31} x]$$

This model has 10 parameters:

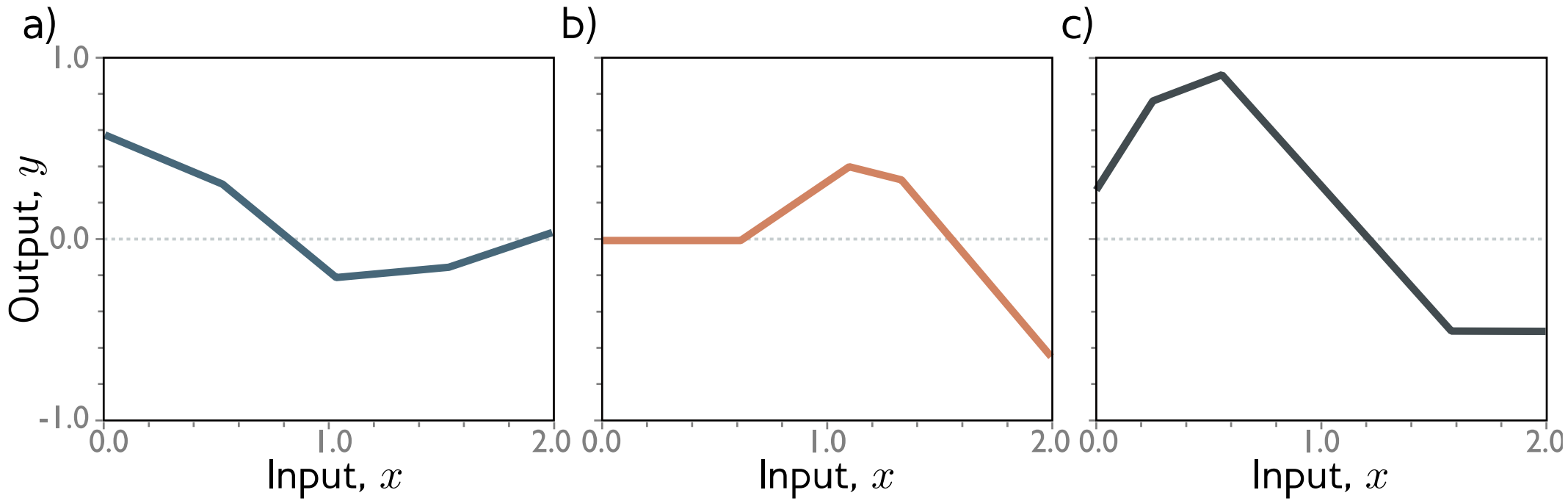
$$\boldsymbol{\phi} = \{\phi_0, \phi_1, \phi_2, \phi_3, \theta_{10}, \theta_{11}, \theta_{20}, \theta_{21}, \theta_{30}, \theta_{31}\}$$

- Represents a family of functions
- Parameters determine particular function
- Given parameters can perform inference (run equation)
- Given training dataset $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}$ it is possible to change parameter to fit a target function

Example Shallow Networks

Given this family of functions, we can compute three specific functions by using different choices for the ten parameters.

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$



Hidden units

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$

Break down into two parts:

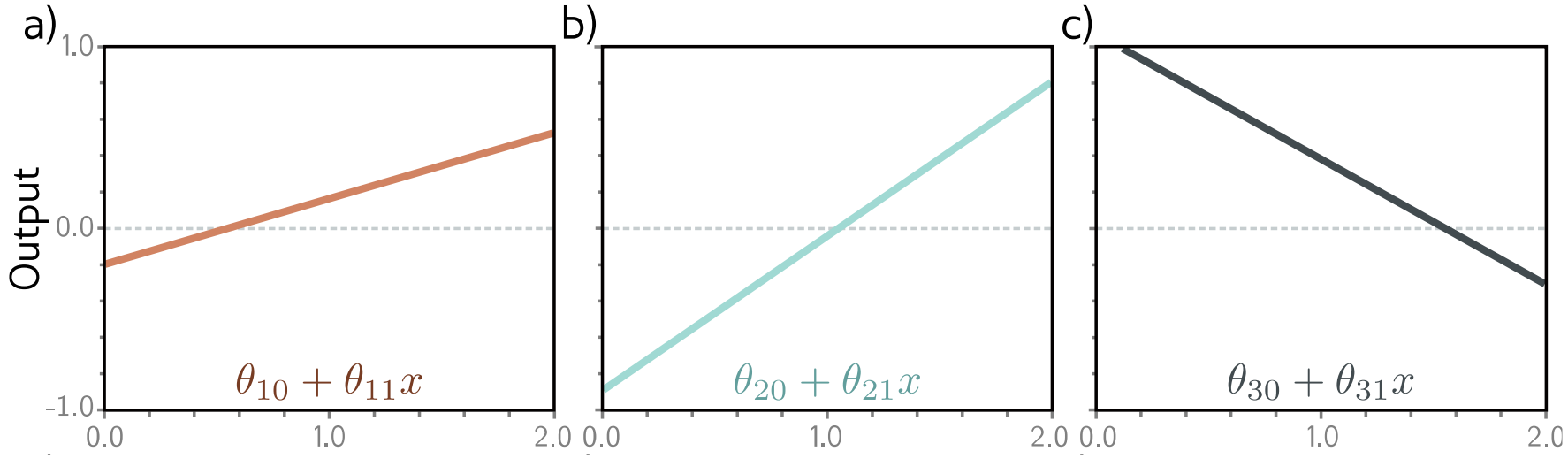
$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

where:

Hidden units

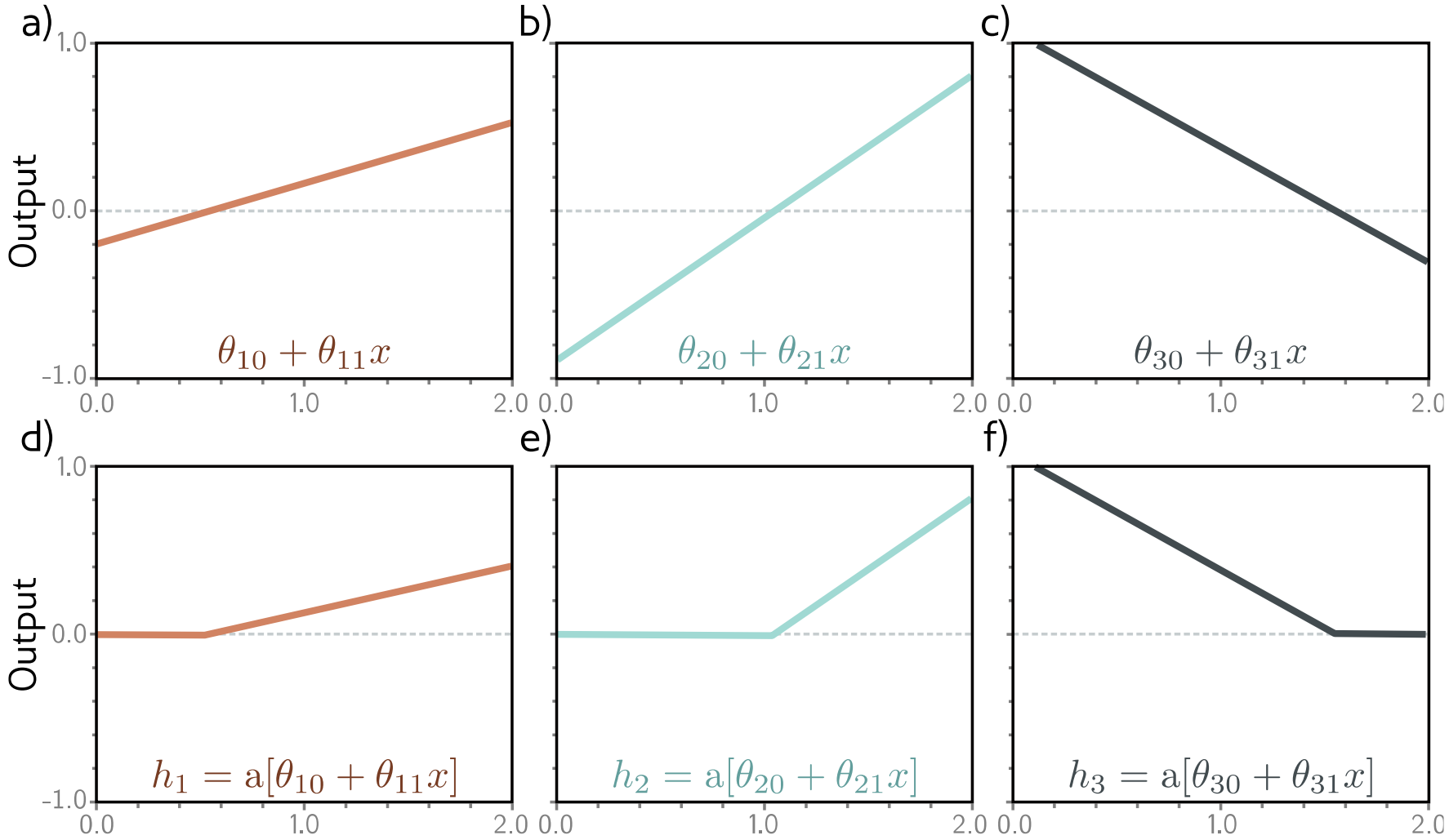
$$\begin{aligned} h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\ h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\ h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x] \end{aligned}$$

1. Compute Three Linear Functions

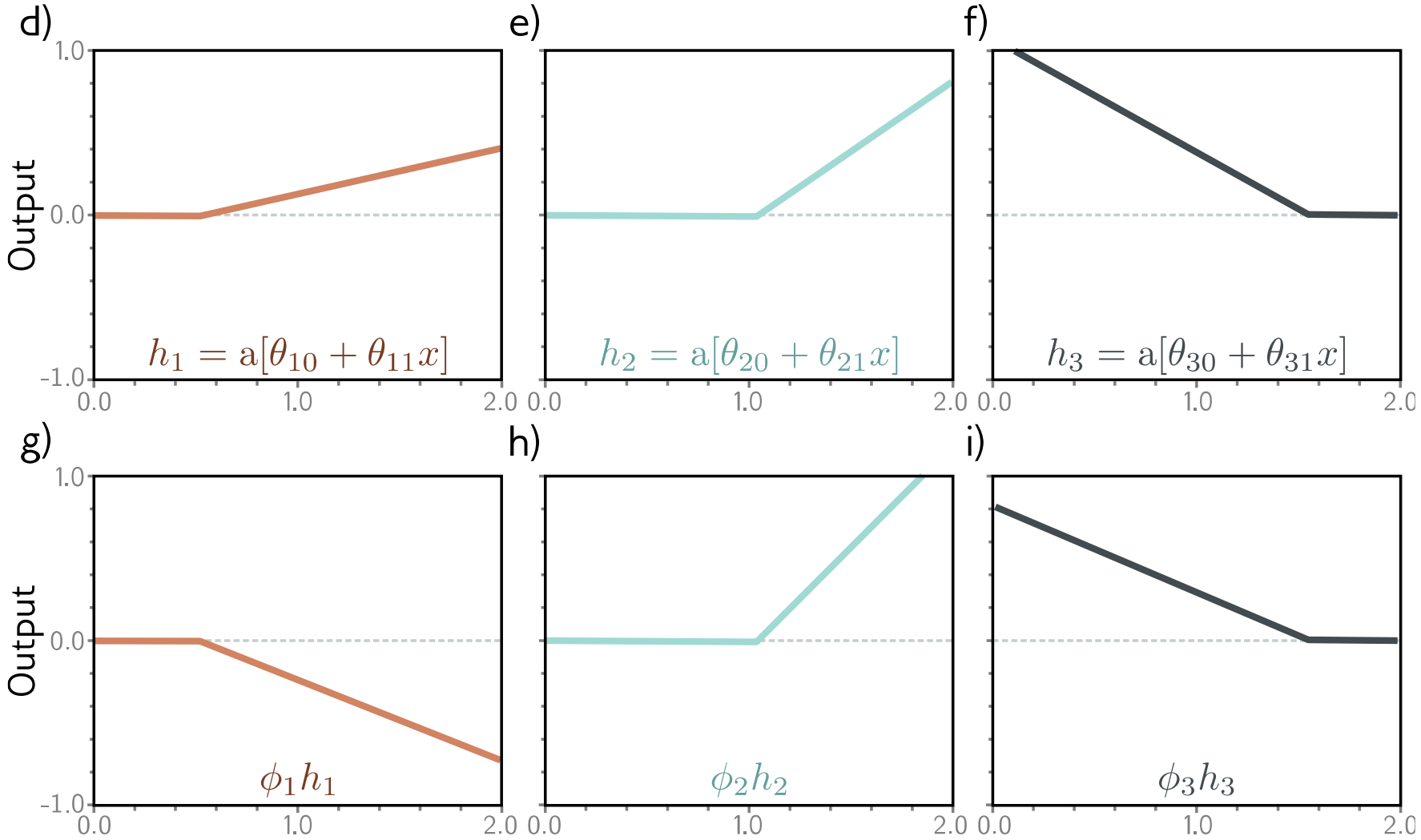


2. Pass through ReLU Fucntions

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x],$$

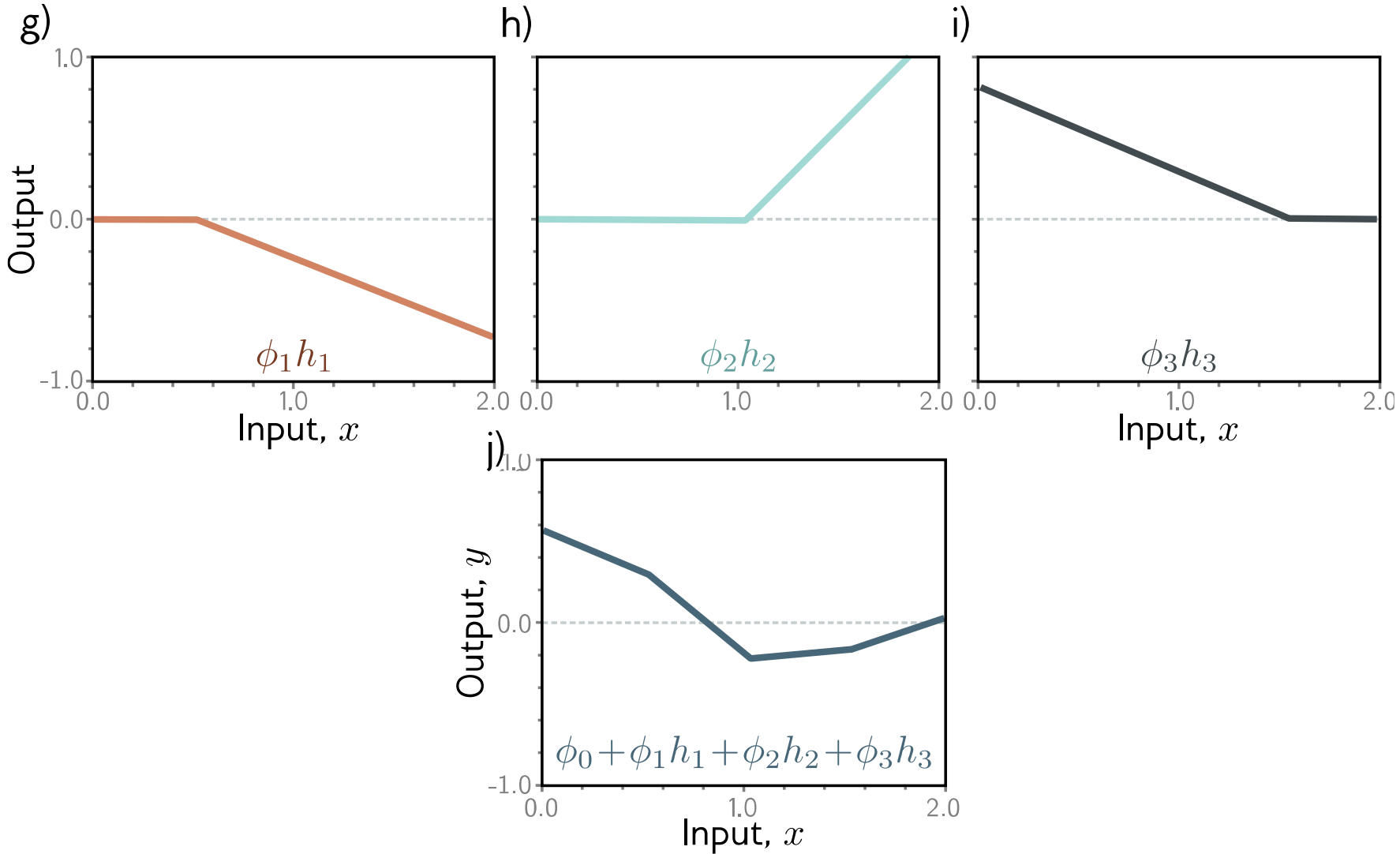


3. Weight the Hidden Units

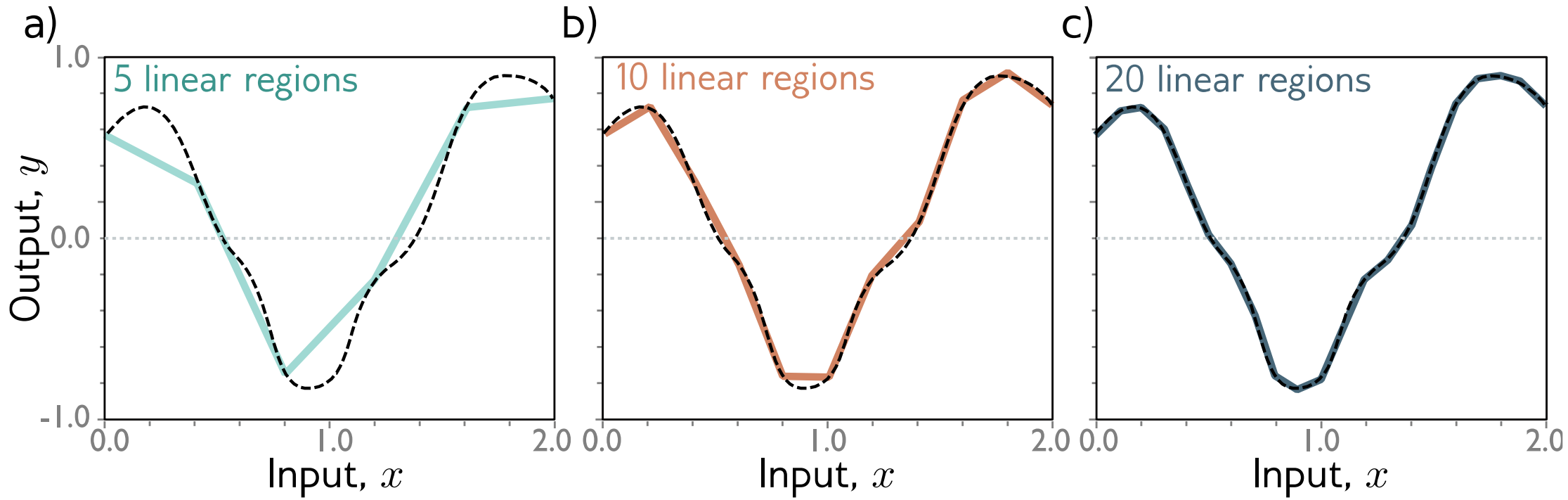


4. Summ the weighted hidden units to create outuput

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$



With enough hidden units



Universal approximation theorem

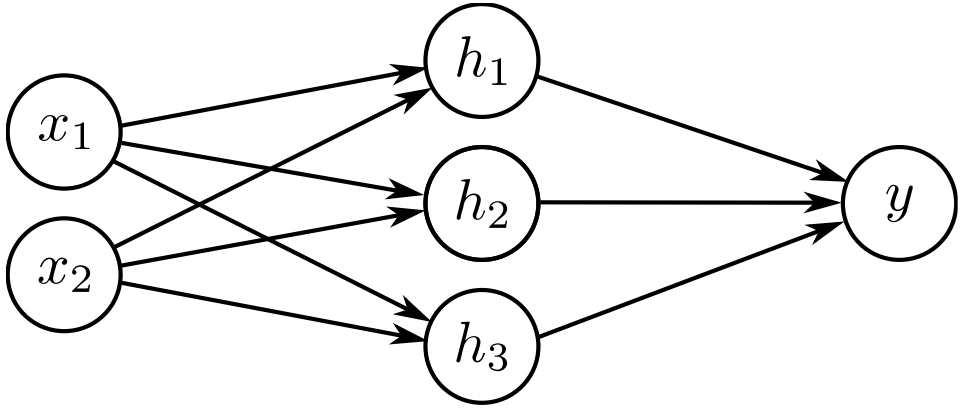
"a formal proof that, with enough hidden units (D), a shallow neural network can describe any continuous function on a compact subset of $\mathbb{R}^D$ to arbitrary precision"

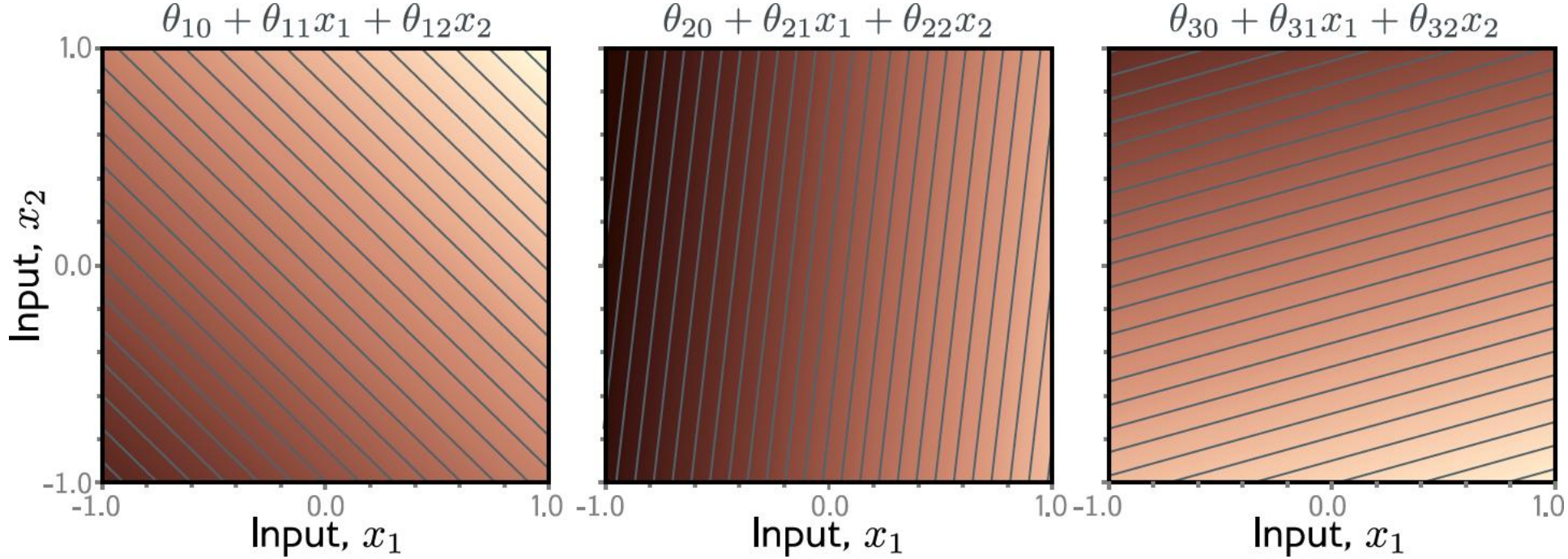
Two Input

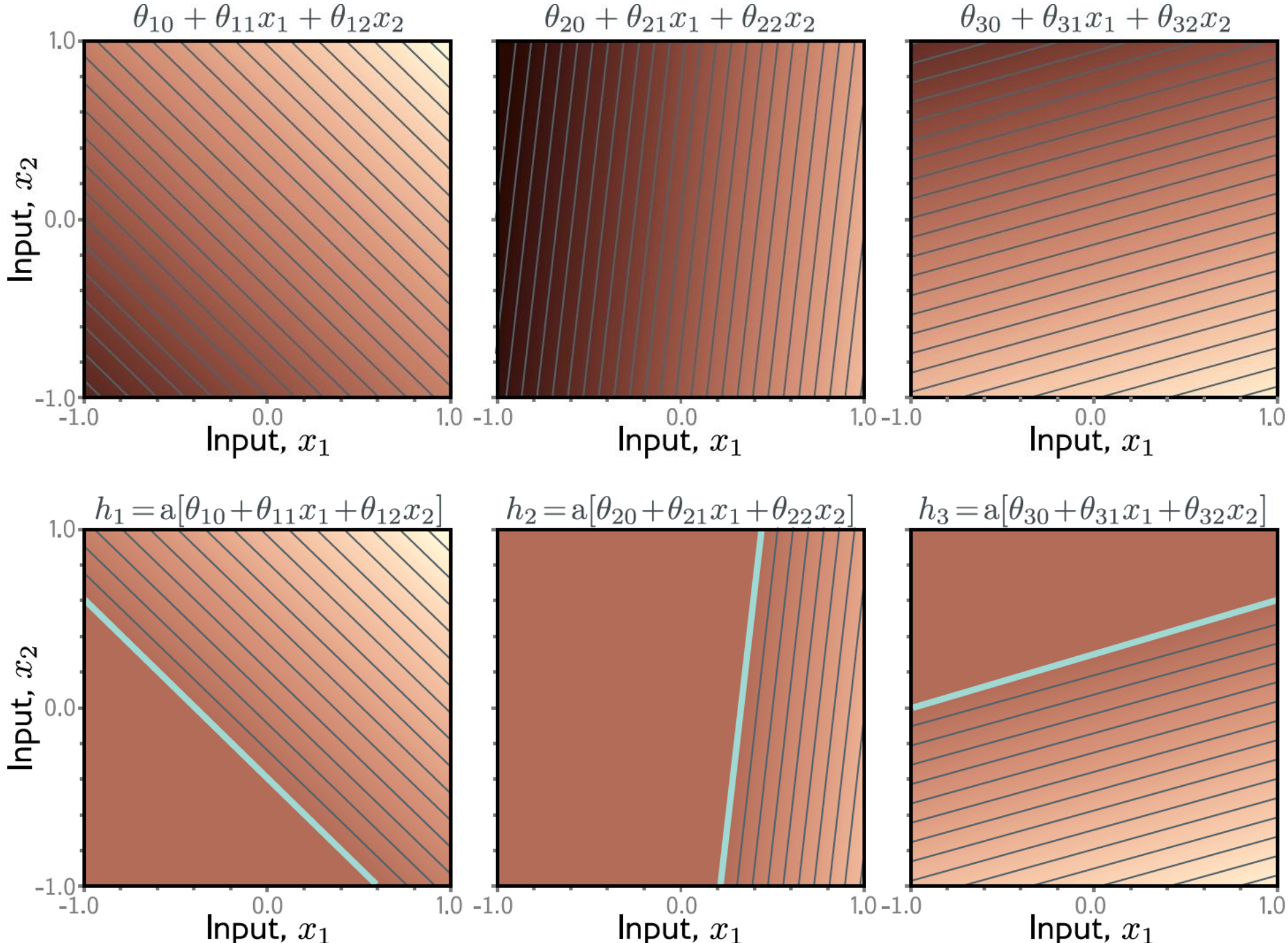
- 2 inputs, 3 hidden units, 1 output

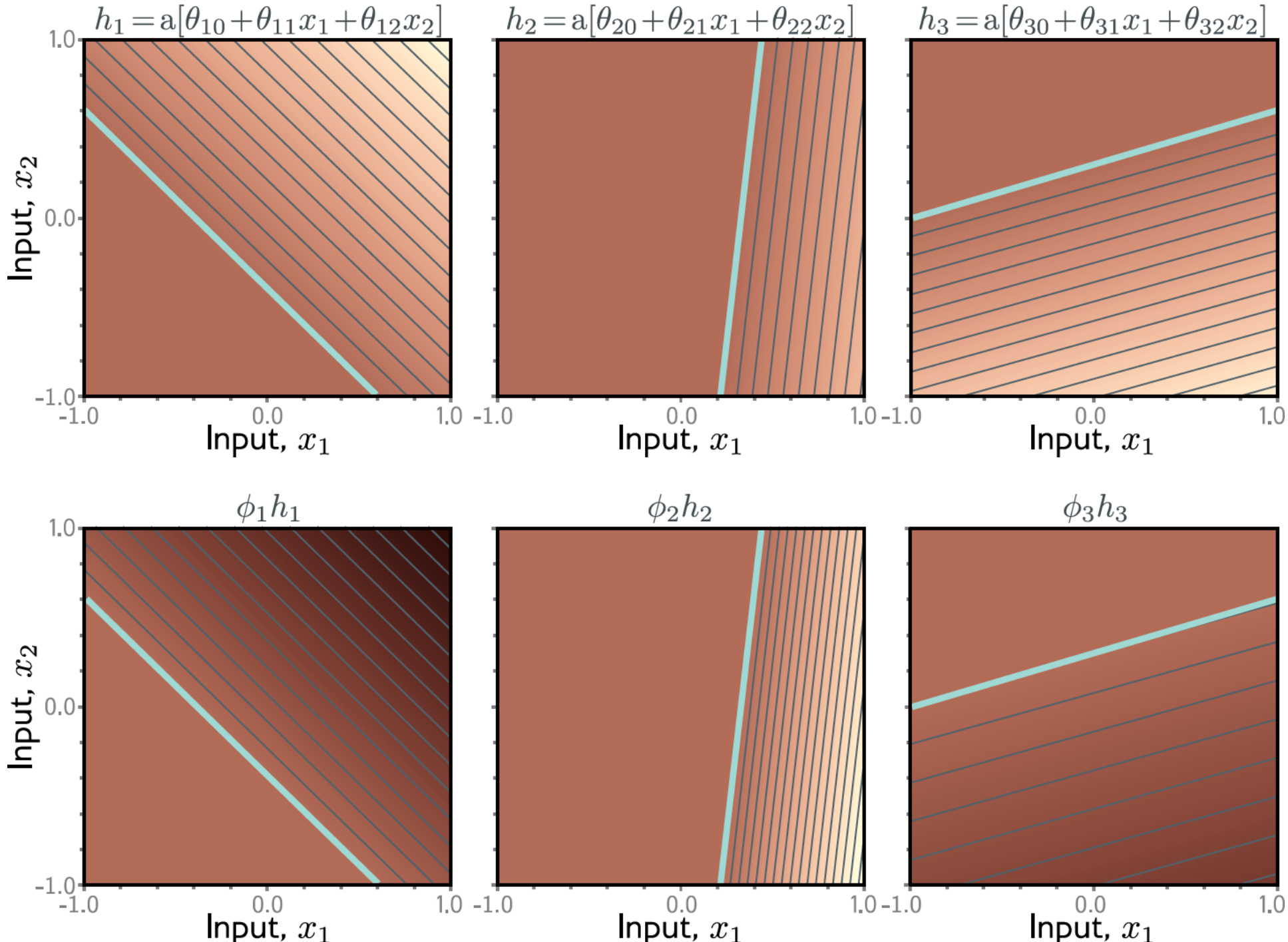
$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$$

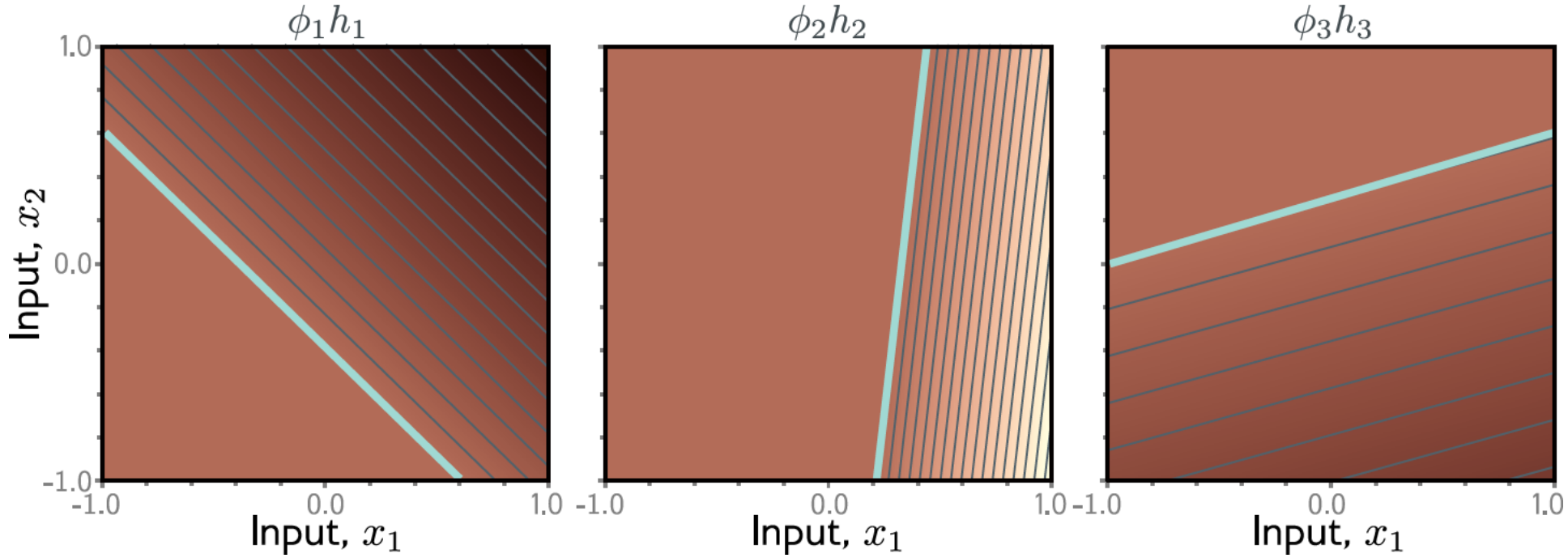
$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

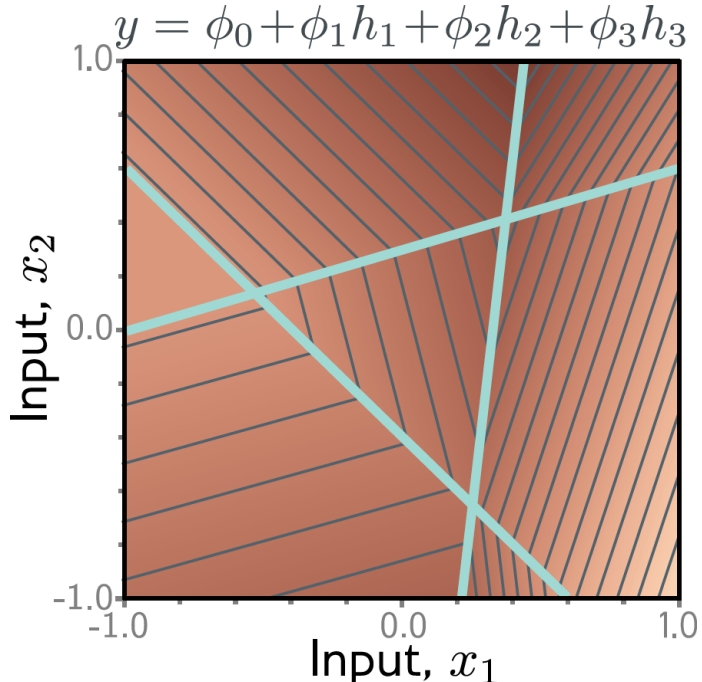


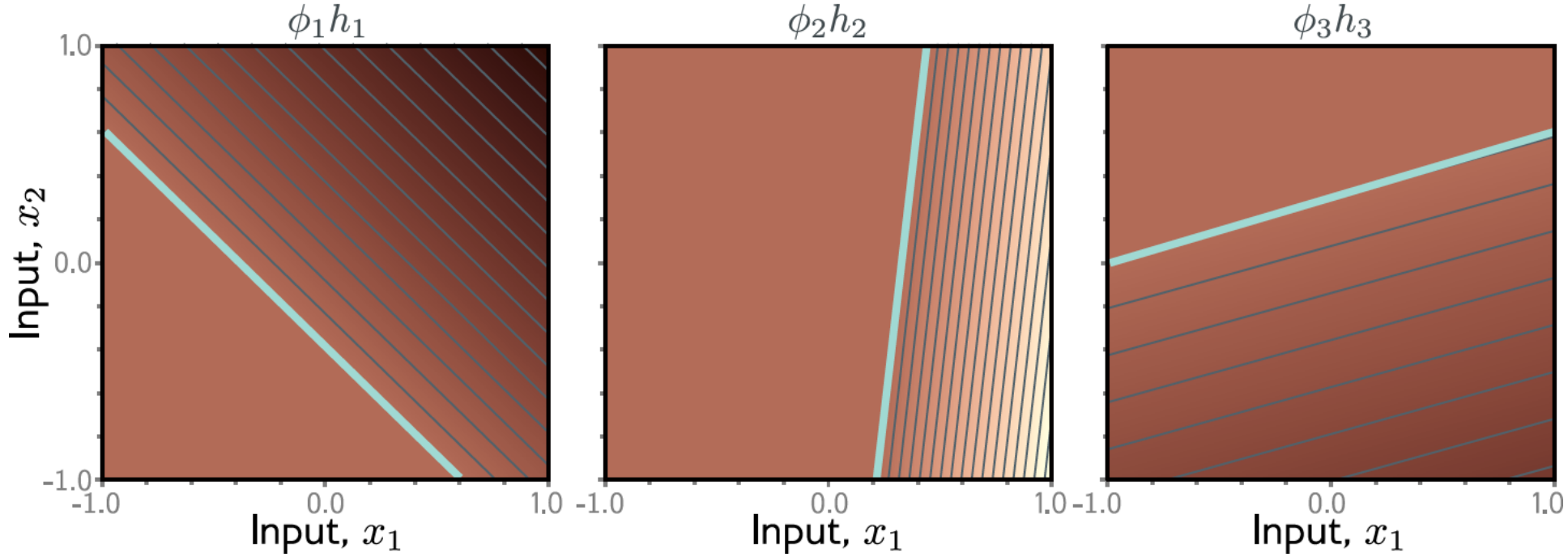

$\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$
$\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2$
$\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2$
Input, x_2
1.0
0.0
-1.0
-1.0
0.0
1.0
Input, x_1
Input, x_1
Input, x_1

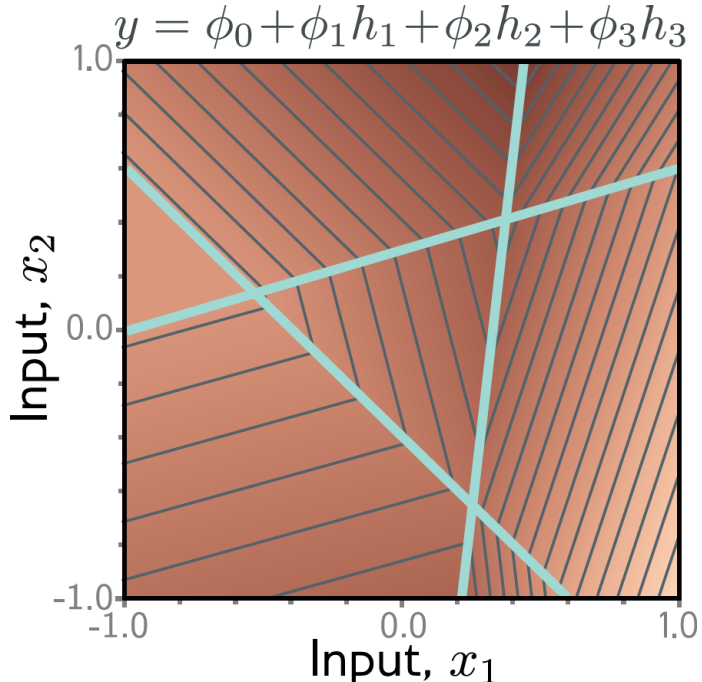

$\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$
$\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2$
$\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2$
$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$
$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$
$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$
Input, x_1
Input, x_2
1.0
0.0
-1.0


$h_1 = a[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$
$h_2 = a[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$
$h_3 = a[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$
$\phi_1 h_1$
$\phi_2 h_2$
$\phi_3 h_3$
Input, x_1
Input, x_2
1.0
0.0
-1.0

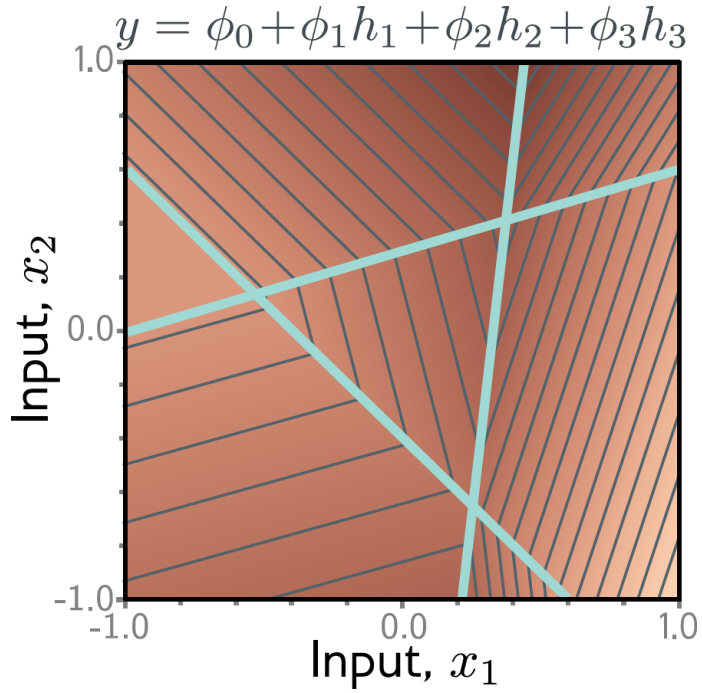

$\phi_1 h_1$
$\phi_2 h_2$
$\phi_3 h_3$
1.0
0.0
-1.0
Input, x_2
Input, x_1

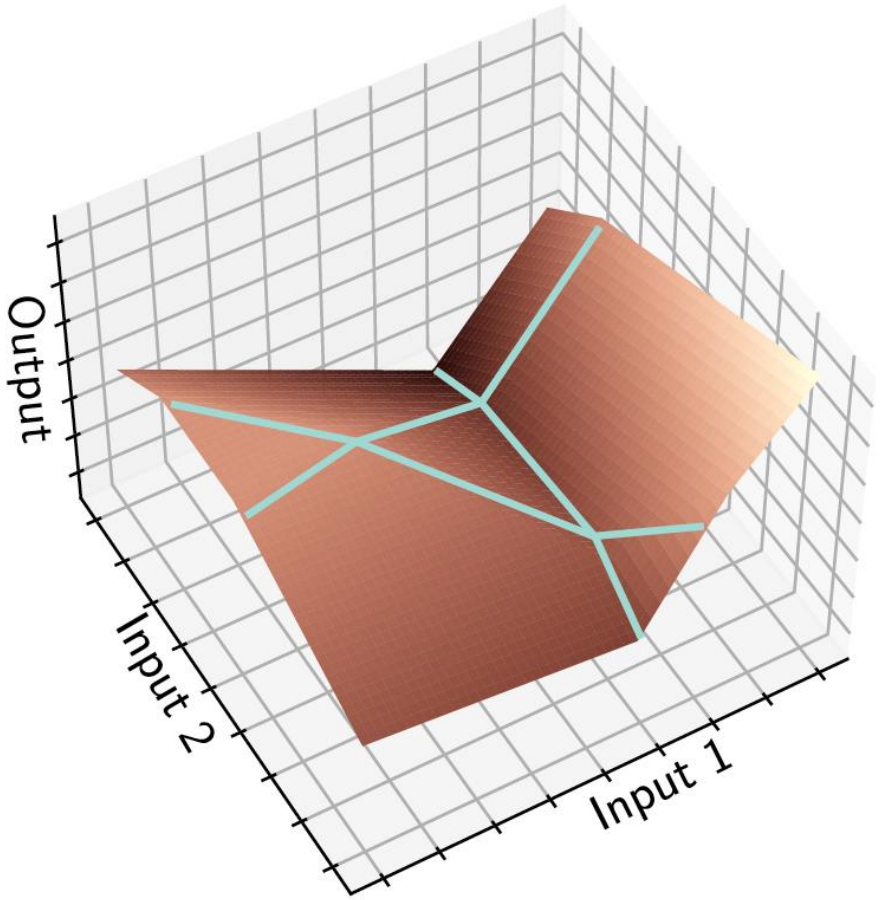

$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$
1.0
0.0
-1.0
Input, x_2
Input, x_1


$\phi_1 h_1$
$\phi_2 h_2$
$\phi_3 h_3$
1.0
0.0
-1.0
Input, x_2
Input, x_1


$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$
1.0
0.0
-1.0
Input, x_2
Input, x_1

Output





Interactive Figures: https://udlbook.github.io/udlfigures/

Shallow vs. Deep Neural Networks

Shallow vs. deep networks

The best results are created by deep networks with many layers.

- 50-1000 layers for most applications
- Best results in
  - Computer vision
  - Natural language processing
  - Graph neural networks
  - Generative models
  - Reinforcement learning

All use deep networks. But why?

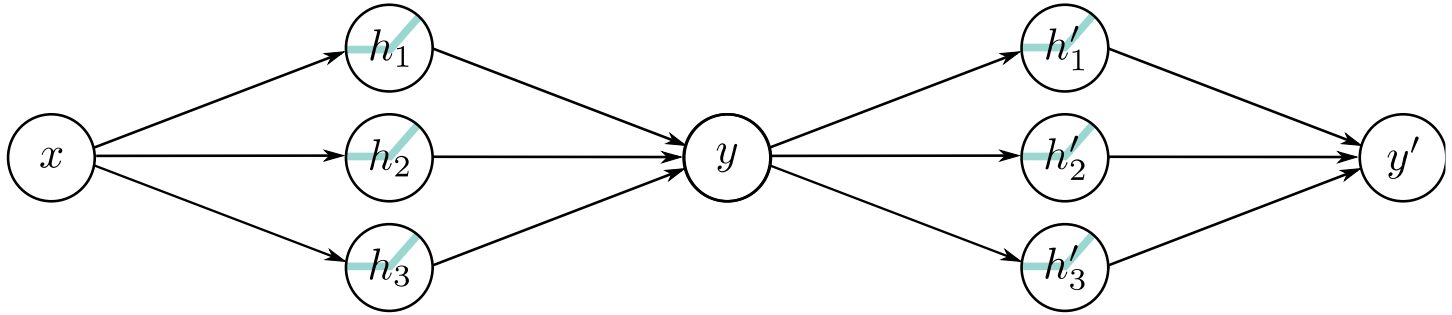
Shallow vs. Deep Learning

Ability to approximate different functions?

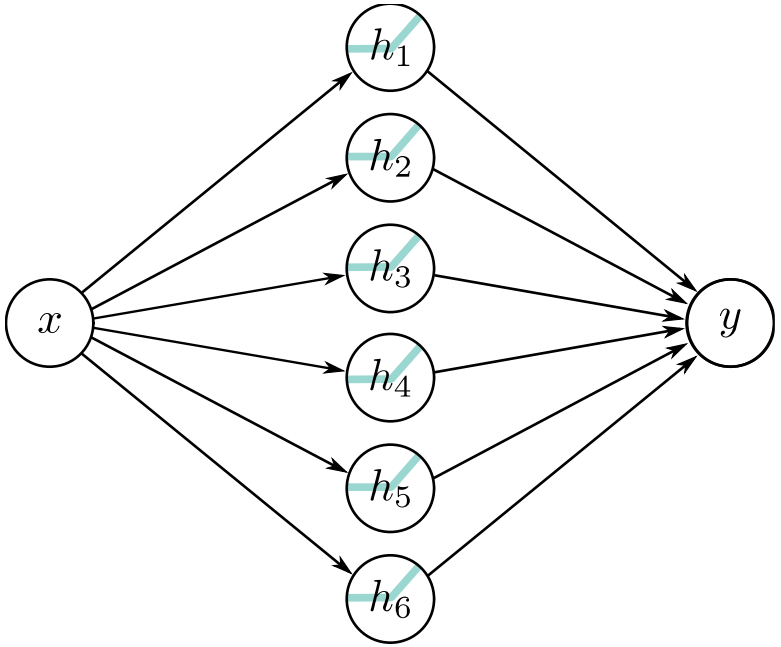
- Both obey the universal approximation theorem.

- Argument: One layer is enough, and for deep networks could arrange for the other layers to compute the identity function.

Comparision with six hidden units



- 20 parameters
- (at least) 9 regions



- 19 parameters
- Max 7 regions

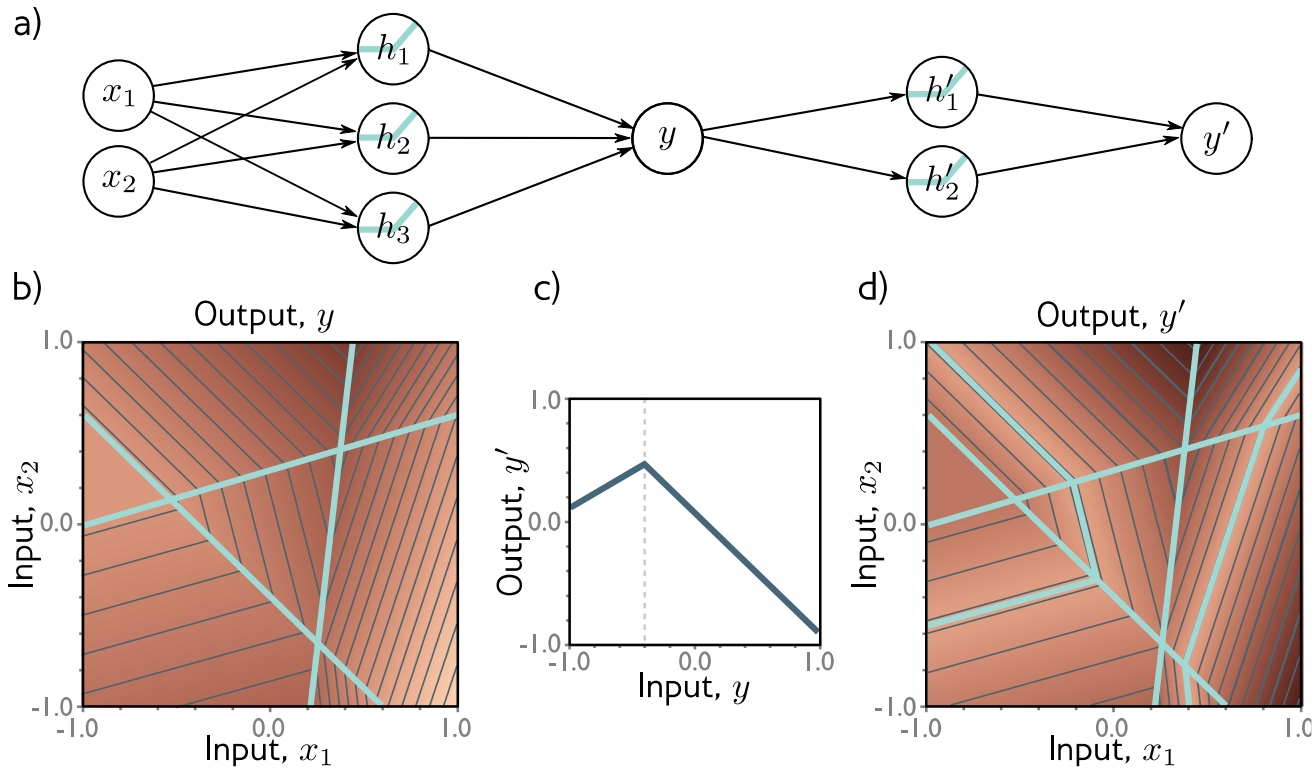
Composing networks in 2D

a) Neural Networks as a composition of two neural networks

b) The first neural network produces a function consisting of seven linear region, one of which is flat.

c) The second network define a function comprising two linear regions

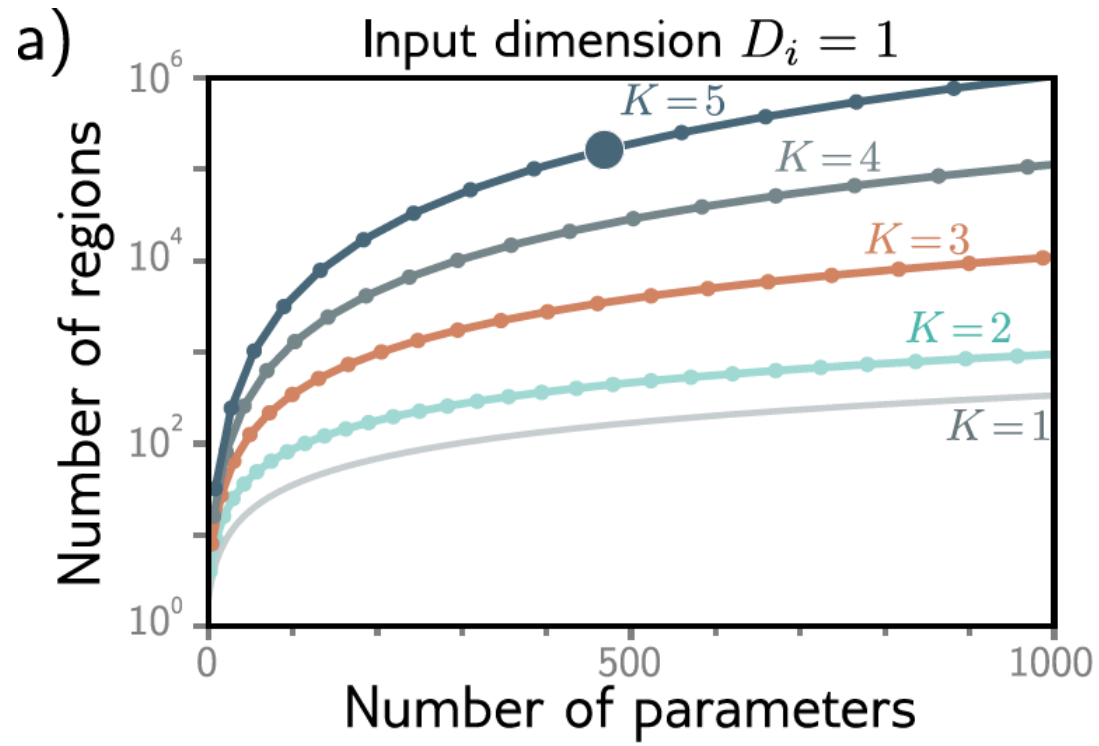
d) When these networks are composed, each of the six non-flat regions from the first network is divided into two new regions by the secondo networks to create a total of 13 linear regions.



Number of linear regions per parameter

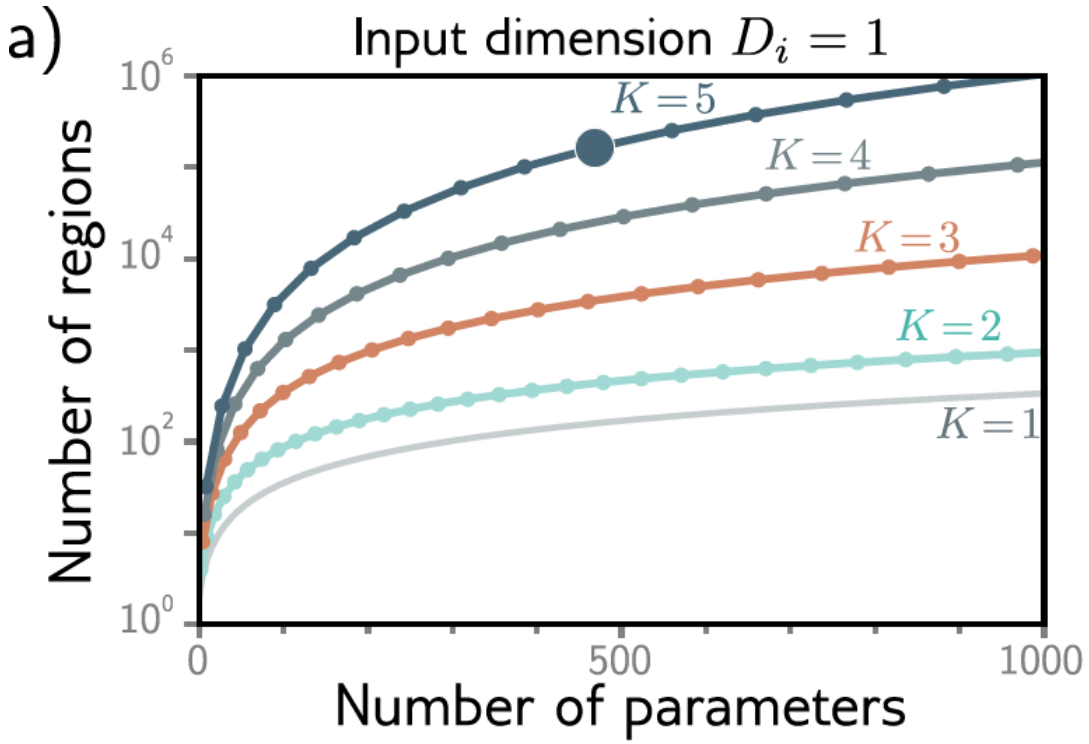
A shallow network with one input, one output, and $D > 2$ hidden units (D= width of network) can create up to $D + 1$ linear regions and is defined by $3D + 1$ parameters.

A Deep network with one input, one output, and K Layer of $D > 2$ hidden units can create a functions with up to $(D + 1)^K$ linear regions using $3D + 1 + (K - 1)D(D + 1)$ parameters.

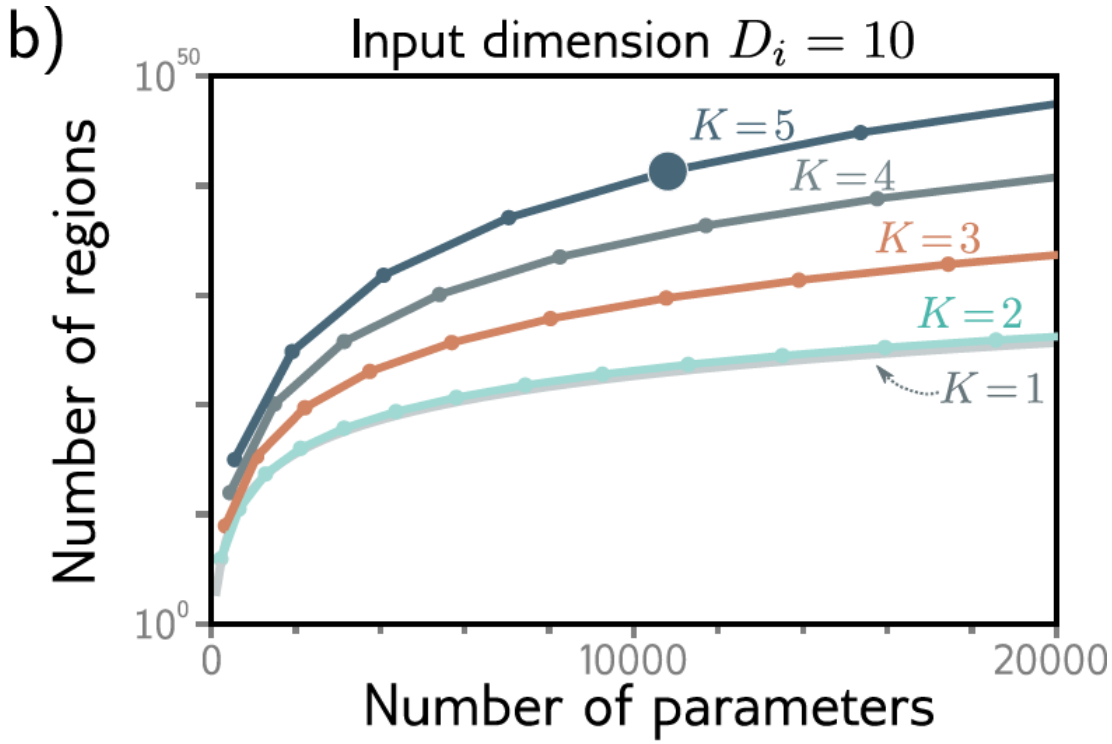


5 layers
10 hidden units per layer
471 parameters
161,501 linear regions

Number of linear regions per parameter



5 layers
10 hidden units per layer
471 parameters
161,501 linear regions



5 layers
50 hidden units per layer
10,801 parameters
$>10^{40}$ linear regions

Fitting

Original MNIST digits

1D Template Patterns

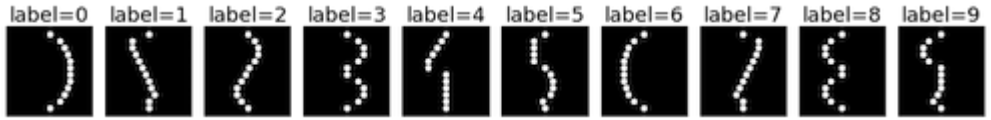
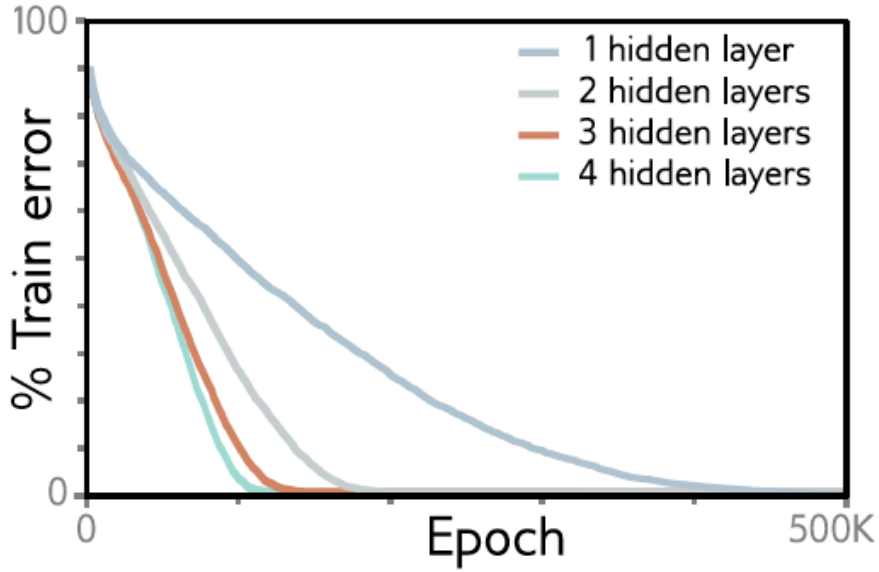


Figure 20.2 MNIST-1D training. Four fully connected networks were fit to 4000 MNIST-1D examples with random labels using full batch gradient descent, He initialization, no momentum or regularization, and learning rate 0.0025. Models with 1,2,3,4 layers had 298, 100, 75, and 63 hidden units per layer and 15208, 15210, 15235, and 15139 parameters, respectively. All models train successfully, but deeper models require fewer epochs.



Suggested Readings

Simon J.D. Price “Understanding Deep Learning”, Chapters 2, 3, and 4 (https://udlbook.github.io/udlbook/)

Quoc V. Le “A Tutorial on Deep Learning – Part1” (see the Material fold)